Change Table:

Date of Change	Related Issue Number	Updated by	Revision
12/6/99	97	C.Collins	Updated section 6.2 Program Naming Standards to include '**B3tsssdd**'.
2/21/00	111, 128	F. Ernst	Added section 1.1.2 to describe the use of diagnostic level in batch programs.
7/18/00	Deliverable Updates for 7/31/00	A. Leaño	Changed logo from HWDC to HHSDC
7/18/00	Deliverable Updates for 7/31/00	A. Leaño	Added standard deliverable introduction.
9/1/00	Deliverable Updates for 7/31/00 Resubmission	R. Frey	
1/12/01	Deliverable updates for 1/12/01 resubmission	J. Speak	Added references to performance region and performance testing.

WELFARE DATA TRACKING IMPLEMENTATION PROJECT

DESIGN/CODING STANDARDS GUIDE

Introduction

Project Description

The Welfare Data Tracking Implementation Project (WDTIP) is an application development project that includes overall project management; designing, building and testing the system; developing and executing user training; communicating with internal and external stakeholders; and deploying the Tracking Recipients Across California (TRAC) Application[1]. In addition, data will be converted from county systems to the TRAC database. It is anticipated that this conversion will entail both automated and manual methods. Subsequent ongoing batch data loads from the counties are also included in the WDTIP. The scope of the project is further discussed in the ***Project Scope*** subsection of this document.

Project Purpose and Objectives

In response to the Personal Responsibility and Work Opportunity Reconciliation Act (PRWORA) of 1996, the State of California passed Assembly Bill (AB) 1542. AB-1542 institutes the TANF program in California and imposes welfare time limits, as well as new programmatic and eligibility rules. In addition to welfare time limits, AB-1542 mandates work requirements through the CalWORKs program. As a result of the CalWORKs program, county welfare departments are required to have a mechanism to track eligibility time limits and other related data on an individual level, across counties, and over time to comply with the tracking requirements of both State and Federal mandates.

The purpose of the WDTIP, therefore, is to provide a communication mechanism and central data repository that can be accessed by all technology-enabled counties and relevant agency systems in order to meet the requirements of SAWS legislation and the TANF and CalWORKs programs. It address the immediate need for Federal and State Welfare Reform tracking requirements imposed by the Federal PRWORA, AB 1542 and relevant All County Letters issued by the California Department of Social Services (CDSS).

To this end, The objectives of the project are to satisfy the aforementioned legislative requirements by providing a statewide repository for welfare reform data elements and to facilitate communication between disparate county welfare and statewide welfare-related systems. The primary data to be collected, calculated (if necessary), and tracked for applicants/ recipients includes:

- TANF 60-Month Clock
- CalWORKs 60-Month Clock
- CalWORKs 18/24-Month Clock

[1] The application was named TRAC per discussions with Project Management on 9/13/99. See Issue #60 of the ***WDTIP Issue Tracking Log***.

- Able Bodied Adult Without Dependents (ABAWD) Calendar

Project Scope

The overall objective of the WDTIP is to provide a communication mechanism and central data repository that can be accessed by all technology-enabled counties and relevant agency systems. In addition, it must enable counties to prevent welfare fraud and meet the requirements of Welfare Reform. The scope of the WDTIP includes design, construction, testing, and implementation of an application that will allow all 58 California counties to accurately track individual welfare recipient information to meet the requirements of both State and Federal welfare reform. The project also consists of the development of CICS screens to view data and approximately 10 operations and management reports. A one-time conversion of county data will be required for the initial load into the database with subsequent ongoing loads performed by counties. Examples of data to be tracked include:

- PRWORA time clock calculation
- CalWORKs time clock calculations, including exceptions and exemptions
- ABAWD status
- Diversion information
- Other tracking to be defined by the WDTIP Joint Requirements Planning Workgroup (e.g., sanctions, childcare, work participation, case participation, etc.)

The conversion of county data to populate the TRAC database will be a vital component of the WDTIP. To this end, the project scope includes the following conversion activities: design, development, testing and implementation of conversion programs, including, but not limited to:

- Identification of required county data elements to populate TRAC
- Identification of county file format requirements
- Development of edit and error processing rules
- Assistance with the one-time initial conversion
- Development of ongoing load requirements for county data into the TRAC
- Guidance for development of extract requirements to county technical resources

In addition, the scope of the project will include the following implementation activities:

- Regional training sessions
- Regional information sessions
- County visits as needed
- Consistent and ongoing communication with stakeholders
- Implementation support

The scope of this project does not include:

- Resources to convert county data into a standard file for conversion and ongoing data loads
- Assisting agencies/counties with the design and development of county welfare system screens to view TRAC data
- Development or management of any changes to the Statewide Client Index (SCI) application

Design/Coding Standards Deliverable Introduction

Document Objective

The Design/Coding Standards Document outlines programming standards that will be followed by the WDTIP development team to successfully implement the WDTIP application. This deliverable breaks down standards into specific development areas to address overall design and coding standards outlined in the Statement of Work.

Document Purpose and Audience

The purpose of the **Design/Coding Standards Document** is to provide developers with a standard method of system design and module development to achieve uniformity across components of the system.

This documentation is meant solely for system developers with a working knowledge of mainframe system operations, COBOL development, JCL, SQL, and development environments. The document should provide a comprehensive description of development standards for technical staff to follow when designing and coding modules.

Document Scope

The **Design/Coding Standards Document** includes the following information:

COBOL Coding Standards - This section outlines coding standards to be followed for all COBOL code generated for the WDTIP application.

JCL Standards - This section outlines coding standards to be followed in developing JCL's for the WDTIP application.

Database Standards for DB2 - This section describes naming conventions to be used for the DB2 database.

SQL Coding Standards - This section outlines SQL coding standards to be followed for all SQL code generated for the WDTIP application.

Library and Naming Standards - This document provides standards for the naming of WDTIP libraries and datasets.

Program and Transaction Naming Standards - This section provides standards for naming transactions and programs.

Screen Standards - This section outlines standards for the design and development of the WDTIP screens.

Report Standards - This section outlines standards for the design and development of the WDTIP reports.

Unit Testing Standards - This section will outline the methodology to be used by individual developers in unit testing modules

Guiding Principles

- WDTIP code shall conform to the design/coding standards outlined in this document.
- Compliance with the standards will enhance productivity by promoting good coding practices.
- Constructions that may increase the likelihood of errors will be identified and avoided.
- All individual modules will be of simple construction and encapsulate limited functionality.
- WDTIP code must be easy to maintain.
- Code reusability must be emphasized.
- Documentation must be an integral part of the development process and not a post-development activity.
- Adequate levels of documentation and comments will be provided to afford succeeding programmers an understanding of historical development.
- These guiding principles will be used as a standard during subsequent code walk-through meetings.

Section I: COBOL Coding Standards

This section outlines coding standards to be followed for all COBOL code generated for the WDTIP application. Machine generated code (e.g., TELON) is exempt from the following standards unless otherwise noted, including associated custom code.

1.1 Identification Division

1.1.1 Identification Sections

The Identification Division will consist of the following information:

Program ID: Refer to **section 6.2** for program naming standards.

Author: The name of the original author of the program.

Date Written: The date on which the program was originally written.

Date Compiled: The date of the last compile (supplied by the system).

Remarks: This section must be designated as a comment - COBOL II does not recognize REMARKS as a reserved word. The REMARKS section will consist of the following:

Program Title: The full name of the program (i.e. “Individual Clearance”).

Program Maintenance Log: This will contain the date that a change was made, the name of the person who made the change, and a description of the change. Below is an example of how the program maintenance log should be formatted.

Date	Analyst(s)	Change Description

Input/Output Files: A list of input and output files utilized by program.

1.1.2 Diagnostic Level

In traditional development, COBOL 'display' statements are used during testing to validate the correct execution of a program and to resolve errors. Normally, but it is not a mandatory requirement, these statements are deleted after the testing process and the code is then moved to the production environment. Should errors occur at a later stage that require this trace facility to be reinserted, these display statements will again have to be added in order to isolate the problem, duplicating time and effort that was already spent during the actual development. However, rather than simply putting in straight display statements, an IF clause could be placed around it, and based on the value of some externally supplied variable, the statement can either be displayed or not displayed. In this way, a 'trace' facility can be developed that does not require any recompilation. The Expeditor trace facility provides a means to navigate through the code and display views to determine problems, however, this requires separate generation activity. Information can be written to SYSOUT, during the running of a batch program to depict the program execution activity, which will hopefully reduce the need for generation with trace. This may be especially useful if later amendments are made to the program and a verification of activity is desired. It is for this reason that a 'diagnostic level' facility will be included in our templates to control trace activity. Throughout the main batch program and subroutines, this diagnostic level will be referenced, and depending on its value, trace information displayed to SYSOUT.

A value will be used, in the range 0 to 5, to determine the level of trace information that will be supplied to the user in the SYSOUT, during the running of the particular batch program.

A broad outline for the diagnostic levels is as follows:

- 0 ***Production Level*** - This is the level of tracing used for production level jobs. No tracing information is provided apart from the checkpoint information taken at the duration or count intervals.
- 1 ***Record Level*** - This level of tracing provides comments at the record or main section level. For example, prior to calling a routine, a message of the 'About to call XXXXXX' type may be displayed.
- 2 ***Paragraph Level*** - This level of tracing provides comments at the called paragraph level to show progress within a called paragraph's execution.
- 3-5 ***Key Data Level*** - These levels of tracing may be to used to provide the values of key data at certain points during an action block's execution, allowing the user to visually trace, in detail, the results of a program run. The user could use this information, for example, to validate the job run output against the contents of external files.

Each lower level of tracing incorporates the previous levels. Therefore, if the diagnostic level is set to 5, all diagnostic statements (0 to 5) will be displayed in SYSOUT. If the diagnostic level is set to 0, only diagnostic statement of 0 will be put into SYSOUT.

'Trace' statements will be incorporated in the main program or subroutines block. The following example displays how the statements will be incorporated:

```
IF diagnostic_level  >= '1'
  DISPLAY "About to Open files"
ENDIF

IF diagnostic_level >= '2'
```

```
  DISPLAY "In Open para"
ENDIF

IF diagnostic_level >= '3'
  DISPLAY ' WTW DATE is " ws-wtw-date
ENDIF
```

1.2 Environment Division

1.2.1 Input-Output Section

This section will contain a FILE CONTROL paragraph with SELECT and ASSIGN clauses for each sequential data set used in the program.

> SELECT: When referencing files from more than one system, each file name must be prefixed with the system mnemonic of the system from which it came. The COBOL name of the file must be used (the name having been developed according to the WDTIP File Naming Standards).
>
> ASSIGN: The Data Definition (DD) name portion of the assigned name will conform to the WDTIP Naming Standards.

1.3 Data Division

1.3.1 File Section

Each File Description (FD) will contain, at a minimum, the COBOL name of the file and the RECORD CONTAINS clause.

The BLOCK CONTAINS clause will only be used for ISAM/VSAM files. If using the BLOCK CONTAINS clause, the appropriate blocking factor must be specified. For non-ISAM files, blocking will be controlled through the JCL.

In addition to the clauses mentioned above, other clauses may be used.

The clauses will be followed by the 01 level record description entry. The full record layout will be in the WORKING STORAGE section.

1.3.2 Working Storage Section

Data Division keyword alignment for occurs, redefines, indexed by, value, and comp will be to first attempt to fit the clause on the same line as the defined element. If the clause is too long for the first line, it will begin on a second line with the keyword.

Maintain a copybook structure for every data file.

Indexing will be used rather than subscripting, since indexing is more efficient.

Minimum and Maximum data values should NOT be hard-coded. They should be defined in working storage as constants.

The first line of Working Storage, for batch programs, will contain a literal that states "(Program ID) WORKING STORAGE STARTS HERE".

Each Working Storage group will contain a 01 level filler with a value clause literal describing the group. Each clause will be contained in a copybook that will be copied into the appropriate group. The copybook names are noted below. See the copybook layouts for the specific contents of each data structure.

WORKING STORAGE data will be grouped into the following categories with the following rules for prefixes:

- The variable names should start with **WS-**
- The Constants in working storage section will start with **C-**

The middle portion of the variable is to be descriptive of the variable's meaning. However, it will be free form in the sense that there will not be a fixed requirement for the number of middle nodes, and it will not be subject to the standard abbreviation constraints used for WDTIP data model element names.

The Working Storage groups will be ordered in the same sequence (alphabetical).

Variables in a Working Storage data group must all begin with the same prefix.

Level numbers should be gapped in increments of five with items of the same level aligned.

All numeric fields involved in arithmetic operations should be signed and packed.

1.3.3 Linkage Section

The Working Storage standards listed above will also apply to the Linkage Section. The variable names should start with **LS-**

Field names and PIC clauses will be the same in calling and called programs.

1.4 Procedure Division

1.4.1 General Procedure Division

An EJECT will be used after each paragraph unless multiple paragraphs can fit on one page. In this case, an EJECT will be placed after the last paragraph that can fit completely on the page.

Attempt to eliminate the internal sort in the program, as well as input/output procedures. DFHSORT in JCL can be used instead to serve this purpose.

Positive conditional statements will replace negative ones (IF NOT). Add Negative Logic if it will eliminate code.

The format of the IF Structure should be to indent the second "IF" under the "ELSE" (on If\Else\If statements). For example:

```
------------------------------------------------
      IF

      ELSE
         IF
------------------------------------------------
```

Use a three-space indentation for "IF". Also, "IF" will start at the beginning of the line and there will be one blank space between "IF" and the statement.

There should be one blank line before an IF statement and one blank line after "END-IF".

Use scope terminators when available.

Commenting will be done for every paragraph that will include business logic when appropriate.

Set the desired return code for every JOB abend.

For batch runs, restart method should be provided.

Literals should not be used. Use 88 Level items and Constants instead.

Indexes, subscripts, and switches will be restricted to a single use.

Only one verb will be used per line.

At least one blank line will be left after paragraph names and before paragraph exit names.

IF statements will not be nested more than four levels deep.

Each ELSE statement will appear on a separate line and will be aligned with its corresponding IF statement. All statements within an IF or ELSE will be aligned.

1.4.2 Paragraphs

Paragraphs should be succinct and must accomplish a specific task or set of closely related tasks. If a paragraph exceeds more than a page (excluding comments), evaluate for the possibility of further decomposition.

Each paragraph will have only one entry point and one exit point.

Each paragraph must be PERFORMed THRU the EXIT.

Each paragraph will return to the module that performed it.

Descriptive comments must be included in logic intensive paragraphs or paragraphs that contain unusual conditions or processing.

Each paragraph name will contain a number. Paragraphs will be organized in ascending order.

Paragraph numbers must be a minimum of three digits in length and will not exceed five digits in length.

The following numbering convention will be used in batch programs to define the paragraphs. This convention will apply to both main programs and subroutines.

Paragraph Processing	Paragraph Number Series
Main Line Control	00000
Initialization	10000
Application Functions	40000
End Of Job	70000
File and Data Base Accessing	80000
Common Routines	90000

Paragraphs will be separated by at least one blank line.

Paragraph size should not exceed approximately one page (in general). A paragraph should not contain only one line of code, unless it is executed more than 3 times.

Number similar work in a series of numbers. For example, within the File and Data Base Accessing (80000) series, group data actions as follows:

- ❑ 81000 series for all reads, selects, and fetches
- ❑ 82000 series for all inserts, updates, writes, and rewrites
- ❑ 83000 series for all deletes
- ❑ 84000 series for all OPEN cursors
- ❑ 85000 series for all CLOSE cursors

Each paragraph's "EXIT" will be numbered and named the same as the paragraph header.

1.4.3 Statements and Styles

The following statements and styles are prohibited:

- ❑ ACCEPT FROM CONSOLE
- ❑ ALTER
- ❑ DISPLAY UPON CONSOLE
- ❑ GO TO DEPENDING ON
- ❑ MOVE CORRESPONDING
- ❑ STOP RUN (use GOBACK)
- ❑ SORT may only be used with approval (see QA Team) and on an exception basis

All files will be closed at the end of the program.

Data items specified in the USING phrase of a subroutine will be listed in the same order in the LINKAGE SECTION of the subroutine.

Common paragraphs will be coded for each READ, WRITE, DB2 Database access, Fatal Error, and ABEND.

Use READ INTO instead of READ when reading sequential files.

Use WRITE FROM when writing to a sequential file.

GO TO can only be used to go to the exit of the paragraph being executed.

Use In-line PERFORMs only when the use of traditional PERFORMs would force the division of a function in a way that it would not normally be divided.

Use redefine clause wherever it is deemed appropriate.

1.5 Valid WDTIP Classwords

The list below contains suggested abbreviations to be used for valid classwords in COBOL.

Classwords	Abbreviation
Address	ADR
Amount	AMT
Code	CD
Counter	CTR - new class word for W-S only
Day	DD
Date	DT
Group	GP
Hours	HRS
Identifier	ID
Indicator	IND
Key	KEY
Month	MM
Name	NAM
Number	NUM
Percent	PCT
Quantity	QTY
Switch	SW
Time	TM
TimeStamp	TS
Text	TXT
Year	YY

Section II: JCL Standards

This section outlines the coding standards to be followed in developing Job Control Language (JCL) for the WDTIP application. All jobs will consist of invoking JCL which call in cataloged procedures (PROCs). The environment related differences will be controlled by passing symbolic parameters from the calling JCLs to the called PROCs. The calling JCLs will be different for each environment while the called PROC will be the same.

2.1 General

The following describes the general standards to be followed while creating a PROC/JCL.

- All production runs must be set up as catalogued procedures.
- Parameters should not be entered manually to a job. The WDTIP batch parameter file should be used instead.
- All production data sets must be catalogued except when a file is sent to an outside agency.
- Use of Generation data groups (GDG's) for physical sequential files are mandatory. The WDTIP System Support group should approve any deviations from this.
- All reports must be spooled to a dataset and a separate print jobstep should be added to print the report.
- All temporary disk files should be deleted and uncataloged after successful completion of the job via an IEFBR14 step.
- All completed jobs will be routed to SAR for storage and distribution.

2.1.1 JCL & PROC Notes

Definition: CALL*ING* procs are known as **JCL** members and CALL*ED* procs are known as **PROC** members.

JCLs and **PROCs** should be named as '***B3efsnnx***' where

B3	- Constant for WDTIP		
e	- Environment		
		D	- Development/Unit Test
		T	- System/Integration Test
		A	- User Acceptance Test

S	- Performance Test
R	- Training
P	- Production

f	- Frequency

D	- Daily
W	- Weekly
M	- Monthly
Q	- Quarterly

s	- Function

C	- County Input File Concatenation
H	- Monthly Report Transmit to Counties
S	- Sort
T	- Transmit Exception File to County
U	- Update
R	- Report
L	- Load
X	- Submit to Internal Reader

Note: Standards for other functions will be created as needed.

nn	- Job number within that sub-system (01 through 99)
x	- Partition indicator for parallel jobstreams

Symbolic parameters must be coded to assign values to:
- ❑ All environment dependent sub-parameters
- ❑ All generation dataset numbers (except when a new generation of the dataset is created).

There should be only one symbolic per line.

2.2 JCL Statements

This section describes the skeleton standards to be followed for JCL statements. There are shells in place to help you get started. They are named as follows:

Development/Unit Test……....B3.DEV.JCLLIB(SAMPLE)
System/Integration Test……..B3.TEST.JCLLIB(SAMPLE)

User Acceptance Test ………B3.ACC.JCLLIB(SAMPLE)
Performance Test ………B3.PERF.JCLLIB(SAMPLE)
Training ………B3.TRNG.JCLLIB(SAMPLE)
Production…...………………B3.PROD.JCLLIB(SAMPLE)

Begin by copying the appropriate member to the member you are creating. There are also corresponding samples of PROCs in:

Development/Unit Test……..B3.DEV.PROCLIB(SAMPLE)
System/Integration Test…….B3.TEST. PROCLIB(SAMPLE)
User Acceptance Test ………B3.ACC. PROCLIB(SAMPLE)
Performance Test ………B3.PERF.PROCLIB(SAMPLE)
Training ………B3.TRNG.PROCLIB(SAMPLE)
Production…...………………B3.PROD. PROCLIB(SAMPLE)

2.2.1 JOB Statement

Please refer to examples of the following statements contained in the sample JCL/PROC's in the above mentioned samples.

- Job name should be the same as the B3.environment.JCLLIB(member).
- Accounting information is variable as follows:

  'V6xxxxnnne'

xxxx	- Application
nnn	- Series within application
e	- Environment

- The submitter's name is required to be coded.
- The first line of JOB statement should only contain job name, job accounting information, and submitters name parameters.
- Always code TIME parameter.
- The MSGCLASS parameter should be coded as MSGCLASS = A for all environments.
- NOTIFY parameter should always be coded as NOTIFY=&SYSUID
- CLASS – Do not code CLASS statements on jobs.
- No commented JOB statements are allowed in any environment.

- ❑ MSGLEVEL is an optional parameter and should be coded as follows so that all statements and messages will print.

  MSGLEVEL = (1,1)

- ❑ REGION parameter must be coded for DB2 batch jobs. The size of the region should be determined in consultation with the DBA group.
- ❑ TYPRUN=HOLD should not be used in any environment.
- ❑ /*ROUTE PRINT B3SAR must be coded in all production jobs.
- ❑ /*ROUTE PRINT DISCARD must be coded in all non-production jobs.

2.2.2 JOBLIB Statement

- ❑ Use B3PD.SDSNLOAD, DISP=SHR for production and training jobs
- ❑ Use B3TD.SDSNLOAD, DISP=SHR for development, testing, performance and acceptance jobs

2.2.3 JCLLIB Statement

A JCLLIB statement should be included, specifying where to find the PROC. It should look like:

```
JCLLIB ORDER=(B3.PROD.PROCLIB)
JCLLIB ORDER=(B3.TRNG.PROCLIB)
JCLLIB ORDER=(B3.ACC.PROCLIB)
JCLLIB ORDER=(B3.PERF.PROCLIB)
JCLLIB ORDER=(B3.TEST.PROCLIB)
JCLLIB ORDER=(B3.DEV.PROCLIB)
```

2.2.4 Online JCL documentation

Online JCL documentation must be coded in all jobs, and in all environments. Job are relative to "CALLING AND CALLED" procs.

Online JCL documentation must be coded after the JOBPARM statement. The following are sections to the online JCL documentation to be utilized appropriately:

<u>ONLINE DOCUMENTATION FOR JOB</u>

1) JOB DESCRIPTION:
   Example: This job prints mailing labels.
2) SETUP INSTRUCTIONS:
   Describes PARMDATA requirements, etc.

3) SCHEDULING INFORMATION:
   Scheduling will be done using **Execution Schedule Processor (ESP)**.
   Describes the frequency of its execution.
4) PREDECESSOR/SUCCESSOR INFORMATION:
   Describes what jobs run directly before and after this job.
5) VALID CONTROL CODES:
   Details about how to handle any valid non-zero return codes.
6) ABEND/RERUN/RESTART:
   Preferably explained by jobstep.
   If using Quickstart, explain as required.
7) OUTPUT HANDLING:
   If output is to be sent out, include:
   - Contact name
   - Phone number
   - Room number
     AND/OR
   File routing information

8) FOLLOW-UP:
   Special instructions for procedures after the end of the job. (e.g. faxing volser numbers and record counts to the tape library)
9) CRITICAL JOB:
   Yes or No.
10 CUSTOMER CONTACT:
   Specify beeper number of individual responsible for functional area.

Online documentation of job steps for CALLED procs:

- Function of each step in the job preceding the EXEC statement of each step.
- Description of COND and PARM parameters for each step of the job.

2.2.5 EXEC PGM Statement

- COND parameter must be coded to execute or bypass the step depending on the results of the previous job step.

- PARM parameter should be avoided as much as possible. If used, it should be passed as a symbolic parameter.

- TIME parameter should be specified if job time is to override.

2.2.6 DD Statement

- DDNAME should be the name assigned in the program (ASSIGN clause)
- DSN should be coded as such:
  B3.JOBNAME.STEPNAME.DDNAME(+1) or (0)
- Report outputs follows report library naming standards.
- SPACE parameter for flat files should be estimated prior to coding JCL.

2.2.7 SYSOUT Statement

- Acceptance and training report files must be DASD GDG's.
- No hard-coded sysout values are allowed in the production, acceptance, or training environments. &SYSDUMP=Y must be used for SYSU dumps in all environments.
- You must use the model DCB to create GDGs in production jobs.
- DCB=(**B3.GDG.MODEL**,…………………...)

Section III: Database Standards for DB2

This section will detail the naming conventions to be used when creating DB2 objects. Directions on the types of objects to be used and the conditions that apply to their use are also included.

3.1 Overview of DB2 at HHSDC

DB2 is loaded on the computers of the Health and Human Services Agency Data Center (HHSDC) as a shared resource for the use of State projects running applications at that facility. Responsibility for the support of DB2 ultimately rests with the technical support staff of the data center. Application specific design and control issues are handled within the user groups of the project.

This split responsibility for DB2 reserves system level issues to the data center. Consequently, system level items will not be discussed in this document, as they are not within the scope of the project.

3.2 DB2 Database Structure and Organization

DB2 databases will be organized using the following hierarchy of objects:

Storage Groups: Defining the physical DASD used to contain the database.
→ Databases: A collection of tablespaces pertaining to a subject area.
  → Tablespaces: A physical storage space containing data in tables.
    → Tables: A physical storage space containing closely related data.
      → Indexes: A list of values that provides improved access to tables.
      → Views: A logical construct that provides alternate access to tables.

All DB2 databases will use storage groups. No other method of DASD allocation will be permitted.

Segmented or partitioned tablespaces are the norm. Placement of multiple tables within a single tablespace of any type must be justified due to the increased maintenance requirements.

Large tablespaces should be used when necessary.

Indexes will be defined to support application access to the data. Secondary indexes will be defined as necessary. The Technical Support group must review all indexing requirements.

Type 2 indexes will be standard.

3.3 DB2 Object Naming Standards

Object	Format	Definition
Storage Group	Gsssennu ex: *GSISP01D*	G → Constant sss → System name (SIS) e → Environment D - Development/Unit Test T - System/Integration Test A - User Acceptance Test S - Performance Test R - Training P - Production nn → Sequence number u → Application D - Data storage I - Index Storage
Database	Dsssenn ex: *DSISP01*	D → Constant sss → System name (SIS) e → Environment D - Development/Unit Test T - System/Integration Test A - User Acceptance Test S - Performance Test R - Training P - Production nn → Sequence number
Tablespace	Sxxxxxxx ex: *SALIEN*	S → Constant xxxxxxx → Abbreviation of the table name
Table Creator	B3sssee n ex: *B3SISTD2*	B3 → Constant sss → System name (SIS) ee → Environment TD - Development/Unit Test TI - System/Integration Test TA - User Acceptance Test TS - Performance Test TR - Training PD - Production n → Sequence number

Object	Format	Definition
Table	SIS_xx…xx ex: *SIS_AGENCY*	SIS → Constant note: system tables do not require prefix xx…xx → Up to 14 characters describing the subject of the table's data
Index Creator	B3sssseen ex: *B3SISTD2*	B3 → Constant sss → System name (SIS) ee → Environment TD - Development/Unit Test TI - System Integration Test TA - User Acceptance Test TS - Performance Test TR - Training PD - Production n → Sequence number
Index	Xxx…xxn ex: *XNONCAPA2*	X → Constant xx…xx → Up to 16 characters duplicating the table name n → Sequence number 0 - Primary Index 1-9 - Secondary Indexes
View	Vxx…xxn ex: *VAGENCY1*	V → Constant xx…xx → Up to 16 characters describing the view n → Sequence number
Plan	B3xxxnne ex: *B3DU101P*	B3 → Constant xxx→ Application nn → Sequence number e → Environment D - Development/Unit Test T - System Integration Test A - User Acceptance Test S - Performance Test R - Training P - Production

Object	Format	Definition
Collection	B3sssenu ex: *B3SISP1B*	B3 → Constant sss → System name (SIS) e → Environment D - Development/Unit Test T - System/Integration Test A - User Acceptance Test S - Performance Test R - Training P - Production n → Sequence number u → Application B - Batch O - Online
Package	B3xxxxxx ex: *B3SISP1B*	B3 → Constant xxxxxxx → Same as program name
Bind Owners	B3ssseee ex: *B3SISDEV*	Defined through RACF B3 → Constant sss → System name (SIS) eee → Environment/Function D - Development/Unit Test T - System/Integration Test A - User Acceptance Test S - Performance Test R - Training P - Production

3.4 DB2 Database Access

User access to the database will be controlled through DB2 privileges. Privileges to access the database will be granted to the user community by the Technical Support group.

Section IV: SQL Coding Standards

This section outlines SQL coding standards to be followed for all SQL code generated for the WDTIP application.

4.1 Overview

Objective
The objective is to provide efficient and consistent access to DB2 tables across subsystems during all phases of application development and implementation.

Benefits

- Improved productivity.
- Common design methodology for review and problem resolution.
- Planned approach to better utilization of resources and optimal performance in the DB2 environment.

The performance of a DB2 application is dependent on several factors. These factors include the type of access strategy selected to fulfill the request, the method used to join one or more tables, any sorting or data manipulation required in processing the request, the type of locking chosen to control the access to data, etc.

The access path to the data in a DB2 database is not controlled by predefined or user specified access paths. Although some control can be obtained by creation of useful indexes, presence of indexes does not ensure that they will be used. The DB2 optimizer during the BIND operation always makes final selection of table access paths.

The database standards presented in this document strive to prevent inefficient usage of DB2, insure against degradation of performance, and maximize concurrency.

IMPORTANT: Please note that information presented in this document is DB2 V5 Release specific.

SQL is a compiled language (not interpretive) when executed in a dynamic and a static environment. Dynamic SQL provides for executing SQL statements from a host language which are not known until the program is executed. As each dynamic SQL must be optimized and bound during program execution, they are not permitted while formulating SQL statements.

A need for dynamic SQL must be discussed with the database support group.

Explicit use of LOCK TABLE statements in host language is not permitted for online application as it adversely affects concurrency. Requirements for using LOCK TABLE statements in batch applications are normally limited to database maintenance utilities, and therefore must be discussed with the database support group.

4.2 Standards for Application Development

4.2.1 INCLUDE SQLCA

This statement is required in all DB2 programs.

```
          +-----------------------------------------------
+
          ¦
¦
          ¦ EXEC SQL
¦
          ¦
¦
          ¦        INCLUDE SQLCA
¦
          ¦
¦
          ¦ END-EXEC.
¦
          ¦
¦
          +-----------------------------------------------
+
```

4.2.2 INCLUDE VIEW/TABLE

This statement must be included for each table/view accessed by the program.

```
          +-----------------------------------------------+
          ¦                                               ¦
          ¦ EXEC SQL                                      ¦
          ¦                                               ¦
          ¦        INCLUDE  table/view name               ¦
          ¦                                               ¦
          ¦ END-EXEC.                                     ¦
```

```
|                                                |
|                                                |
+------------------------------------------------+
```

4.2.3 DECLARE CURSOR (INQUIRY)

This statement is required for each cursor used in the program and must be placed before it is first opened.

```
+-----------------------------------------------+
|                                               |
| EXEC SQL                                      |
|         DECLARE CUR_nn_cursorname CURSOR      |
|         (WITH HOLD)                           |
|         FOR                                   |
|            SELECT col1,                       |
|                   col2                        |
|            FROM                               |
|                  tablename                    |
|            WHERE                              |
|                   inquiry-predicate           |
|           (OPTIMIZE FOR n ROWS)               |
|          (FOR FETCH ONLY)                     |
| END-EXEC.                                     |
|                                               |
+-----------------------------------------------+
```

nn - is the serial number given to the cursor in the application.

WITH HOLD option - does not close the cursor after the current unit of work is committed. The commit operation in conjunction with WITH HOLD option will release locks that are not required to maintain the cursor. An initial FETCH is required before a positioned update or delete statement can be executed. After the initial FETCH, the cursor is positioned on the row following the one it was positioned on before the commit operation.

IMPORTANT: The option WITH HOLD is to be used only in batch applications as it is ignored in message driven programs such as CICS applications.

OPTIMIZE FOR n ROWS tells DB2 that at most, a total of n integer rows are to be retrieved from the result table. It is recommended to use OPTIMIZE clause when estimates of rows are required to be retrieved by the query are known. This could lead to better performance. The OPTIMIZE FOR clause does not alter the result table or the order in which the rows are fetched. Any number of rows can be fetched, but after a total of n integer fetches, performance can possibly degrade.

FOR FETCH ONLY declares that the result table is for read only. Some result tables are read-only because their nature precludes the use of delete or update operations. In such instances FOR FETCH ONLY is redundant. In other cases specifying FOR FETCH ONLY can possibly improve the performance of FETCH operation.

IMPORTANT: FOR FETCH ONLY cannot be used in a statement containing an UPDATE clause.

4.2.4 DECLARE CURSOR (UPDATE)

A cursor FOR UPDATE OF must be used when the row fetched is to be updated or deleted. Only the column specified in the FOR UPDATE OF list can be updated by UPDATE WHERE CURRENT OF clause. The fetched row can also be deleted using a WHERE CURRENT OF clause.

```
+-----------------------------------------------+
|                                               |
| EXEC SQL                                      |
|         DECLARE CUR_nn_cursorname CURSOR      |
|         FOR                                   |
|            SELECT col1,                       |
|                   col2                        |
|            FROM                               |
|                   table/view name             |
|            WHERE                              |
|                   col1 = :host-var            |
|            FOR UPDATE OF                      |
|                   col2                        |
| END-EXEC.                                     |
|                                               |
| EXEC SQL                                      |
|         UPDATE                                |
|              table/view name                  |
|         SET                                   |
|              col2     = :host-var-nam         |
|         WHERE                                 |
|              CURRENT OF CUR-nn-cursorname     |
| END-EXEC.                                     |
|                                               |
+-----------------------------------------------+
```

nn - is the serial number given to the cursor in the application.

IMPORTANT: DISTINCT cannot be used in a statement containing an UPDATE clause.

4.2.5 OPEN CURSOR

The statement opens a cursor to allow rows to be fetched from its result table.

```
+-----------------------------------------------+
¦                                               ¦
¦ EXEC SQL                                      ¦
¦        OPEN CUR_nn_cursorname                 ¦
¦ END-EXEC.                                     ¦
¦                                               ¦
+-----------------------------------------------+
```

nn - is the serial number given to the cursor in the application.

4.2.6 FETCH

The FETCH statement positions the cursor on the desired row and retrieves information into the host variable structure.

```
+-----------------------------------------------+
¦                                               ¦
¦ EXEC SQL                                      ¦
¦        FETCH CUR_nn_cursorname                ¦
¦         INTO  host-variable-structure         ¦
¦ END-EXEC.                                     ¦
¦                                               ¦
¦                                               ¦
+-----------------------------------------------+
```

nn - is the serial number given to the cursor in the application.

host-variable-structure must be a host structure generated by DCLGEN. This avoids coding host variables with incompatible data types/length in the application program.

4.2.7 CLOSE CURSOR

The statement closes a cursor.

```
+-----------------------------------------------+
¦ EXEC SQL                                      ¦
¦        CLOSE CUR_nn_cursorname                ¦
¦ END-EXEC.                                     ¦
```

```
                +-----------------------------------------------+
```

nn - is the serial number given to the cursor in the application.

IMPORTANT: Explicitly closing cursors as soon as possible improves performance.
CLOSE does not cause commit or rollback operation.

4.2.8 SELECT (ZERO/SINGLE/EXISTENCE)

When zero or one row of information is expected, then this format must be used. In cases where mere existence needs to be checked, then COUNT(*) or EXISTS must be used.

```
+-----------------------------------------------+
¦                                               ¦
¦ EXEC SQL                                      ¦
¦        SELECT                                 ¦
¦              col1,                            ¦
¦               col2,                           ¦
¦        INTO                                   ¦
¦             :host-var1,                       ¦
¦             :host-var2                        ¦
¦        FROM                                   ¦
¦             table/view name                   ¦
¦        WHERE                                  ¦
¦             search-condition                  ¦
¦ END-EXEC.                                     ¦
¦                                               ¦
+-----------------------------------------------+
```

IMPORTANT: Using SELECT and checking for SQLCODE of -811 (duplicate rows) is not permitted.

Presence of an Index will have to be ensured before using COUNT(*).

4.2.9 SELECT * (ALL COLUMNS)

SELECT * is not permitted in a SQL statement.

It detracts from data independence.

If a column is added to a table, the host language will not be prepared to accept the additional column.

If a column is deleted from a table, the host variable will not match the received columns.

4.2.10 UPDATE (MASS/SEARCHED)

When multiple rows are updated with a single SQL statement, they consume unpredictable amount of resources (making it difficult to tune the system) and must therefore be avoided in the online application program.

When multiple rows are to be updated a cursor must be opened with FOR UPDATE OF clause and must be used to update the information.

```
+-----------------------------------------------+
¦                                               ¦
¦ EXEC SQL                                      ¦
¦         UPDATE                                ¦
¦               table/view name                 ¦
¦         SET                                   ¦
¦                col-name = :host-var-nam       ¦
¦         WHERE                                 ¦
¦                CURRENT OF CUR-nn-cursorname   ¦
¦ END-EXEC.                                     ¦
¦                                               ¦
+-----------------------------------------------+
```

Batch applications, which require mass updates to be made in a single SQL statement, must be discussed with the database support group. This helps in scheduling and estimating the duration of the batch window.

4.2.11 UPDATE OF COLUMNS USED IN WHERE CLAUSE

DB2 will not use an index in an UPDATE statement, when the column (if part of an index) specified in the WHERE clause is also specified in the SET clause.

Also, in a cursor operation, DB2 will not use an index on the basis of a column that is specified in FOR UPDATE OF.

Example: If an index is present on the col2, DB2 will not use the index if the column is part of the SET clause of the UPDATE statement.

```
+------------------------------------------------+
¦                                                ¦
¦ EXEC SQL                                       ¦
¦        UPDATE                                  ¦
¦              table/view name                   ¦
¦        SET                                     ¦
¦               col1     = :host-val1            ¦
¦               col2     = :host-val2            ¦
¦        WHERE                                   ¦
¦               col2 = :host-val3                ¦
¦ END-EXEC.                                      ¦
¦                                                ¦
+------------------------------------------------+
```

4.2.12 DELETE (MASS/SEARCHED)

When multiple rows are deleted with a single SQL statement, they consume an unpredictable amount of resources (making it difficult to tune the system) and must therefore be avoided in the online application program.

When multiple rows are to be deleted then a cursor must be opened with FOR UPDATE OF clause and must be used to delete the information.

```
+--------------------------------------------+
¦                                            ¦
¦ EXEC SQL                                   ¦
¦        DELETE FROM                         ¦
¦              table/view name               ¦
¦        WHERE                               ¦
¦              CURRENT OF CUR-nn-cursorname  ¦
¦ END-EXEC.                                  ¦
¦                                            ¦
+--------------------------------------------+
```

Batch application that requires mass deletes to be made in a single SQL statement must be discussed with the database support group. This helps in scheduling and estimating the duration of the batch window.

4.2.13 DISTINCT

This statement returns an unique row among a set of multiple rows having same column values that are selected. Execution of this statement causes an internal sort, but it is still preferable to selecting all the rows and eliminating unwanted rows by the application program. The presence of an index on the columns that are candidates for sort will yield faster results.

```
+------------------------------------------------+
|                                                |
| EXEC SQL                                       |
|         DECLARE CUR_nn_cursorname CURSOR       |
|          FOR                                   |
|             SELECT                             |
|                     DISTINCT col1,             |
|                              col2              |
|                                                |
|              FROM                              |
|                     view/tablename             |
|             WHERE                              |
|                      search-condition          |
| END-EXEC.                                      |
|                                                |
|                                                |
|                                                |
+------------------------------------------------+
```

nn - is the serial number given to the cursor in the application.

4.2.14 ORDER BY

A SQL statement using ORDER BY invokes an internal sort to arrive at an intermediate result table that consists of the rows of the table. ORDER BY is the only way to insure the sequence of retrieved rows within a cursor. Even when the data in a table is indexed, its retrieval sequence will be unpredictable without the ORDER BY clause.

4.2.15 GROUP BY

A SQL statement using GROUP BY invokes an internal sort to arrive at an intermediate result table that consists of a grouping of the rows of the table.

<u>IMPORTANT:</u>

- ❑ Select only required columns when a sort is required (ORDER BY, GROUP BY, DISTINCT and UNION) to minimize the resources required for the sort.
- ❑ SQL statements containing ORDER BY or GROUP BY must be carefully examined to avoid cost overruns in terms of resources consumed. Presence of suitable indexes/organization of the data helps to alleviate the problems ensuring a better response time during execution.
- ❑ The requirement of ORDER BY / GROUP BY will be reviewed by the database support group.

4.2.16 LIKE

A SQL statement using LIKE causes the rows to be retrieved in which the value (pattern) of the column mentioned in the predicate matches with the host-variable.

<u>IMPORTANT:</u> Predicates using LIKE in a host variable do not use matching index scan and hence can affect response time adversely. The need for LIKE must be discussed with the database support group so that alternate ways can be explored such as usage of BETWEEN instead of LIKE.

4.2.17 BETWEEN

The BETWEEN predicate determines whether a given value lies between two other given values specified in ascending order.

```
+------------------------------------------------+
¦                                                ¦
¦ EXEC SQL                                       ¦
¦         DECLARE CUR_nn_cursorname CURSOR       ¦
¦         FOR                                    ¦
¦            SELECT                              ¦
¦                             col1,              ¦
¦                             col2               ¦
¦                                                ¦
¦            FROM                                ¦
¦                   view/tablename               ¦
¦            WHERE                               ¦
¦                   col1 BETWEEN :host-var1 and  ¦
¦                                :host-var2      ¦
¦                                                ¦
¦ END-EXEC.                                      ¦
+------------------------------------------------+
```

nn - is the serial number given to the cursor in the application.

The values limiting the search are inclusive.

If the search boundary values are known use of BETWEEN is recommended.

IMPORTANT: Using col1 NOT BETWEEN or NOT col1 BETWEEN does not use a matching index scan and can adversely affect response time. Such usage must be avoided.

4.2.18 IN

The IN predicate is equivalent to separate OR conditions for each of the values listed.
The IN predicate compares a value from a collection of values.

```
+------------------------------------------------+
¦                                                ¦
¦ EXEC SQL                                       ¦
¦         DECLARE CUR_nn_cursorname CURSOR       ¦
¦         FOR                                    ¦
¦            SELECT                              ¦
¦                   col1,                        ¦
¦                    col2                        ¦
¦                                                ¦
¦            FROM                                ¦
¦                   view/tablename               ¦
¦            WHERE                               ¦
¦                    col1 IN ( :host-var-list)   ¦
¦                                                ¦
¦                                                ¦
¦ END-EXEC.                                      ¦
¦                                                ¦
+------------------------------------------------+
```

nn - is the serial number given to the cursor in the application.

IMPORTANT: Using the predicates in the following way must be avoided, as it cannot use an index:

col1 IN (subquery)
col1 NOT IN (subquery)

Using the IN predicate in the following way does not use the matching index scan and must be discussed with the database group to explore other alternate ways:

col1 NOT IN (list)
NOT col1 IN (list)

4.2.19 SUBSELECT

A subselect is a subset of a fullselect.

Subquery: a select that forms part of a predicate.

Correlated Subquery (Reference): A reference to a column in a subquery of a table identified at a higher level is called correlated reference.

IMPORTANT: The usage of Subquery or correlated subquery is not recommended and must be discussed with the database group to explore alternate ways to resolve application requirements.

```
+-----------------------------------------------+
¦ ======>>>> example of a subquery <<<<======    ¦
¦ EXEC SQL                                       ¦
¦         DECLARE CUR_nn_cursorname CURSOR       ¦
¦         FOR                                    ¦
¦            SELECT                              ¦
¦                    col1,                       ¦
¦                   col2                         ¦
¦                                                ¦
¦            FROM                                ¦
¦                   view1/table1                 ¦
¦             WHERE                              ¦
¦                    col1 IN ( SELECT col-view2  ¦
¦                              FROM              ¦
¦                                  view/table2)  ¦
¦ END-EXEC.                                      ¦
¦                                                ¦
+-----------------------------------------------+
```

4.2.20 UNION v/s MULTICOLUMN "OR" PREDICATES

All queries containing multicolumn 'OR' predicates must be reformulated using "UNION". This allows DB2 to use an index if available.

An internal sort will be invoked when "UNION" is used to eliminate duplicate rows. If the two sets of information are disjoined or if the application can handle duplicates then "UNION ALL" must be considered. This will eliminate the sort.

```
+-----------------------------------------------+
¦ EXEC SQL                                      ¦
¦         DECLARE CUR_nn_cursorname CURSOR      ¦
¦          FOR                                  ¦
¦             SELECT                            ¦
¦                    col1,                      ¦
¦                    col2,                      ¦
¦                    col3                       ¦
¦              FROM                             ¦
¦                    view/tablename             ¦
¦             WHERE                             ¦
¦                     col1 IN ( :host-var-list) ¦
¦                 OR                            ¦
¦                     col2 = :host-var-value    ¦
¦ END-EXEC.                                     ¦
¦                                               ¦
+-----------------------------------------------+
```

nn - is the serial number given to the cursor in the application.

The previous query can be reformulated to effectively use an index as follows:

```
+-----------------------------------------------+
¦ EXEC SQL                                      ¦
¦         DECLARE CUR_nn_cursorname CURSOR      ¦
¦          FOR                                  ¦
¦            SELECT                             ¦
¦                   col1,                       ¦
¦                   col2,                       ¦
¦                   col3                        ¦
¦            FROM                               ¦
¦                   view/tablename              ¦
¦            WHERE                              ¦
¦                    col1 IN ( :host-var-list)  ¦
¦            UNION                              ¦
¦            SELECT                             ¦
¦                   col1,                       ¦
¦                   col2,                       ¦
¦                   col3                        ¦
¦             FROM                              ¦
¦                   view/tablename              ¦
¦            WHERE                              ¦
¦                    col2 = :host-var-value     ¦
¦ END-EXEC.                                     ¦
+-----------------------------------------------+
```

nn - is the serial number given to the cursor in the application.

4.2.21 AVOID NUMERIC CONVERSION

The data type of a column and its associated host variable used in the WHERE condition must be the same. In addition, it must have the same scale and precision. This problem is resolved if the DCLGEN variables are used.

4.2.22 AVOID ARITHMETIC EXPRESSIONS

Using arithmetic expressions in the 'WHERE' clause must be avoided. No indexes are used when arithmetic expressions are placed in the 'WHERE' clause. The arithmetic must be performed in COBOL prior to execution of the query.

4.2.23 AVOID CONCAT / CONCATENATION OPERATOR

CONCAT or concatenation operator, || must be avoided. No indexes are used when concatenation operators are used in predicates. Need for concatenation operations must be discussed with the database support group.

4.2.24 SCALAR FUNCTIONS

SUBSTR OPERATOR

Need for SUBSTR operator in predicates must be discussed with the database support group.

4.2.25 "NOT" PREDICATE

A predicate with a "NOT" gets converted to its equivalent (whenever possible: example - "NOT >" is changed to "<="). However it is not possible (and therefore no indexes will be used) when the following predicates are used:

- ❑ NOT BETWEEN
- ❑ NOT LIKE
- ❑ NOT IN
- ❑ NOT =

The above restrictions must be noted while formulating SQL in the application.

4.2.26 COMMIT/ROLLBACK

A COMMIT/ROLLBACK statement is not permitted for an online transaction. DB2 will handle the unit of work termination upon the completion of the program. The COMMIT/ROLLBACK should be used in batch application programs where multiple units of work are processed.

```
+-----------------------------------------------+
¦ EXEC SQL                                      ¦
¦          COMMIT WORK                          ¦
¦ END-EXEC.                                     ¦
¦                                               ¦
¦ EXEC SQL                                      ¦
¦          ROLLBACK WORK                        ¦
¦ END-EXEC.                                     ¦
+-----------------------------------------------+
```

4.2.27 COLUMN FUNCTIONS

Column functions MIN, MAX, AVG, SUM in embedded SQL can result in a NULL VALUE being returned to the program. This condition is reached when there are no rows selected by the query predicate or if the table is empty.

The application program must incorporate a null indicator and check for the value in the null indicator. If the null indicator returns a negative value then the host-variable contains null value. Checking the null indicator and taking appropriate actions will avoid program abends during program execution.

```
+-------------------------------------------------+
¦ EXEC SQL                                        ¦
¦         SELECT MAX(SEQ_NO)                      ¦
¦         INTO                                    ¦
¦                :HOSTVAR :NULLIND                ¦
¦         FROM   T0341_IN_DEMOGRAPH               ¦
¦ END-EXEC.                                       ¦
¦                                                 ¦
¦ if nullind < 0                                  ¦
¦    hostvar contains NULL value.                 ¦
¦                                                 ¦
+-------------------------------------------------+
```

The WDTIP standard for date field is DB2 ISO format. The WDTIP standard high value date is '9999-12-31'. The WDTIP standard low value date is '0001-01-01'. The DB2 standard default value for date field in tables is the current date.

4.2.28 JOIN OPERATIONS

Retrieval of rows from DB2 tables via join must be discussed with the database support group. Careful analysis will be given to the predicate selection of the join in order to maximize statement efficiency.

Outer joins are discouraged in batch or online programs. They are very resource intensive and difficult to tune. If a requirement is identified which seems to require an outer join, discuss the requirement with the database support group.

4.3 RELATED STANDARDS FOR APPLICATION DEVELOPMENT

4.3.1 CONDITIONAL STATEMENT - WHENEVER

The conditional statement WHENEVER must not be used in the application program as it does not follow structured programming practices. Instead the returned SQLCODE must be explicitly checked and appropriate actions must be taken. Control must be passed on to the standard abend routine in case hard errors are encountered.

```
+-----------------------------------------------+
|                                               |
| EXEC SQL                                      |
|         INSERT                                |
|              INTO table-name                  |
|                  (col1,col2)                  |
|             VALUES                            |
|                  (val1,val2)                  |
| END-EXEC.                                     |
|                                               |
| EVALUATE SQLCODE                              |
|     WHEN 0                                    |
|          CONTINUE                             |
|     WHEN -803                                 |
|          duplicate row logic                  |
|     WHEN OTHER                                |
|          standard abend logic                 |
| END-EVALUATE.                                 |
+-----------------------------------------------+
```

The example given above assumes that there is a unique index defined on the table.

4.3.2 DB2 ERROR HANDLING

All DB2 errors that cannot be handled within the logic of the program will be passed to a subroutine for processing. The subroutine will be provided by the Technical Support group and will use logic provided by DB2 for decoding SQL return codes and SQL state variables.

4.3.3 COBOL BIND PARMS

All DB2 programs will use package binds. Program binding in the development environment should use the following parameters:

```
DSN SYSTEM(B3TD)
        BIND PACKAGE(B3xxxxxx)          -
        OWNER(B3xxxxxx)                 -
        QUALIFIER(B3xxxxxx)             -
        MEMBER(B3xxxxxx)                -
        ACTION(REPLACE)                 -
        CURRENTDATA(YES)                -
        DEGREE(1)                       -
        DYNAMICRULES(BIND)              -
        ENABLE(*)                       -
        EXPLAIN(YES)                    -
        FLAG(W)                         -
        ISOLATION(CS)                   -
        RELEASE(COMMIT)                 -
        SQLERROR(NOPACKAGE)             -
        VALIDATE(BIND)
END
```

Variables to the BIND parameters will be supplied by the Technical Support group.

Section V: Library and Naming Standards

This section provides standards for the naming of WDTIP libraries and datasets.

5.1 WDTIP Library Names

Proposed Libraries

Below is a list of proposed libraries with the following middle level qualifiers:

DEV	Development/Unit Test
TEST	System/Integration Test
ACC	User Acceptance Test
PERF	Performance Test
TRNG	Training
PROD	Production

LIBRARY TYPE	NEW LIBRARY NAME	USES
COBOL Source	B3.xxxx.SRCLIB	Source programs
COPYBOOKS	B3.xxxx.COPYLIB	COBOL source /CICS maps
JCL	B3.xxxx.JCLLIB	Job statements
JCL Procs	B3.xxxx.PROCLIB	JCL Procedure library
Control Cards	B3.xxxx.CNTLLIB	JCL Control card library
PROGRAM LOADs	B3.xxxx.LOADLIB B3.xxxx.CICS.LOADLIB	(For Batch) (For Online)
TABLE Copybooks	B3.xxxx.DCLGEN	Declaration generator generated copybooks for DB2 tables
CICS MAPS	B3.xxxx.MAPLIB	Defines screen appearance

** Separate libraries are maintained for reports, depending on the requirement.

5.2 Dataset Naming Standards

There should be a system qualifier, subsystem if necessary, indicator for REPORT/DATA/CONTROL FILE.

The Datasets created from JCL should follow the following naming standard:

B3.JOBNAME.STEPNAME.DDNAME(n)

n Applicable only if the dataset is GDG.

Section VI: Program and Transaction Naming Standards

This section provides standards for naming transactions and programs. Specifically, it will describe:

1. Transaction Codes
2. Program Naming Standards

6.1 TRANSACTION NAMING STANDARDS

The trans-id should be defined in such a way that it should represent the application and the hierarchy of the process. Trans-id is limited to 4 bytes .

6.2 PROGRAM NAMING STANDARDS

'**B3tssnnn**' should be the naming standard format for the program where

B3	constant
t	type
	- B for batch
	- C for Online
	- R for report
ss	for subsystem/short description
nnn	number(Should end with zero)

'**B3ssssnn**' should be the naming standard format for the subroutine where

B3	constant
ssss	for subsystem/short description
nn	number (should not end with zero)

'**B3tsssdd**' should be the naming standard format for the copybook where

B3	Constant
t	Type A-agent Y-copybook
sss	Subsystem/short description First 's' should be a 'C' for common subroutines.
dd	Number or Characters depending on if it's an agent or copybook Agents should not end in zero.

Copybooks should end with
WS for working storage
PD for procedure division

EXAMPLES:
Agent: B3AGT01
Regular Copybook (working storage): B3YLDXWS for LoaD eXtraction file WS working storage
Regular Copybook (procedure div): B3YLDTPD for LoaD Table PD procedure division
Common Copybooks: B3YTSQWS OR B3YCOMPD

6.3 SUBSYSTEM IDENTIFIERS

A two-letter abbreviation followed by a two-digit level qualifier will be used in the transaction and program names to identify a subsystem. If multiple levels do not exist within a subsystem, then the two-letter abbreviation will be followed by "00" as the two-digit level qualifier. For example:

Subsystem	Identifier
Application Entry	AA00

Section VII: Screen Standards

This section outlines standards for the design and development of the WDTIP screens.

7.1 Screen Format

All screens in the WDTIP system will adhere to a standard format. The standard format consists of mandatory fields that must be present on all screens in the assigned positions and some optional fields which, if present, must be defined in the assigned position.

The standard screen format is shown below with a description of fields following:

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

*********************************************************************************
01* ++++                    SCREEN TITLE                            >>>>>>>>>>*01
02*                                                                  >>>>>>>>>*02
03*                                                                           *03
04*                                                                           *04
05*                                                                           *05
06*                                                                           *06
07*                                                                           *07
08*                                                                           *08
09*                                                                           *09
10*                                                                           *10
11*                                                                           *11
12*                                                                           *12
13*                                                                           *13
14*                                                                           *14
15*                                                                           *15
16*                                                                           *16
17*                                                                           *17
18*                                                                           *18
19*                                                                           *19
20*                                                                           *20
21*>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>   >>>>>*21
22*>>>> <<<<<<<<<                                                             *22
23*>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>*23
24*>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>*24

*********************************************************************************
   ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+8
```

Legend:	> = Output only field
	< = Input only field
	+ = Input/Output field

The following is a description of the mandatory and optional fields in the standard screen format:

7.1.1 Screen Identification / Tran-ID

Mandatory: All screens will have a four character "screen-Identification/Tran-ID" that indicates the transaction code that is to be used to access that screen. This field will be Alpha/Numeric. This transaction code will allow the user to directly invoke transactions, whose IDs are known to him or her, (except for certain screens which has to be entered in a particular sequence) without having to go through the intervening menu screens.

Starts at column 2 on row 1.

Color: Blue (protected)	Intensity: Normal
Green (unprotected)	Intensity: Normal

7.1.2 Screen Title

Mandatory: All screens will have a title that will be centered on row 1.

Color: Blue Intensity: Normal

7.1.3 Date

Mandatory: The system date will be displayed in the "MM/DD/YYYY" format and will be 10 characters long.

Starts at column 69 on row 1.

Color: Blue Intensity: Normal

7.1.4 Time

Optional: The system time will be displayed in the "HH:MM:SS" format and will be 8 characters long. The display format is military time.

Starts at column 73 on row 2.

Color: Blue Intensity: Normal

7.1.5 PF Keys

PF keys specific to a screen will be shown on row 23 and 24 (if necessary) as literals. When a screen's data continues on to multiple pages, PF keys will be specified in order to scroll the data.

Starts at row 23 and 24.

Color: Blue Intensity: Normal

7.1.6 Key Fields

Mandatory: This will be used only for the screens that correspond to a particular CIN (Client Index Number – key field). The input field varies on a screen by screen basis. The CIN will be the key field. Once the record has been retrieved, this field will become clear. For rest of the screens, this will be clear (including the literal).

7.1.7 More…Indicator

Optional: When a program encounters a situation that requires it to display more data that can fit on one screen, it will set the "MORE" indicator on the screen. This indicates to the user that they can scroll forward to view the additional data. Such screens will have a "MORE..." indicator field that displays the literal "MORE..." if there is more data to view, but will be blank otherwise.

Starts at column 75 on row 21

Color: White Intensity: High

7.1.8 Error/Informational Messages

Mandatory: There will be an error/informational message line to display error messages and/or other informational messages. The error message line will have the format '9999 - XXXXXXXXXXXXXXXXX...' where 9999 is the error message number and XXXXXXXXXXXXXXXXX... is the actual message. The error message should be retrieved from the table.

Starts at column 2 on row 21.

Color: White Intensity: High

7.1.9 Screen Fields

Input fields: All modifiable data fields will appear as underscores so as to indicate the entire field size.

Example: First Name: _______________

Inquiry Fields: All inquiry fields will have unused character spaces filled with spaces.

Example: First Name: PATRICIA

Standard Data Fields: All standard information will be displayed as entered on the screen. Data entered in upper and lower case will be displayed in upper case.

Example:	Date:	MM DD YYYY
	Phone #:	999 999 9999
	SSN#:	999 99 9999

Amount Fields: All amount fields must be entered with dollar and cents value when a decimal point is used or just the dollar value if no decimal point is used.

Example:	If $400.00 is entered as:	400.00__
	It will be displayed as:	__400.00
	If $400.00 is entered as:	400_____
	It will be displayed as:	__400.00

7.2 Screen Colors and Intensity

The following are the standard colors and intensity attributes to be followed for the WDTIP screens.

1. The color and intensity for the standard fields will be as described in the Standard screen format section above.

2. The color and intensity for the other fields on the screen will be as per the table below.

Field Type	Protection, Intensity & Color
Literals	Protected, Normal, Blue
Modifiable data fields	Unprotected, Normal, Green
Modifiable data fields in error	Unprotected, High, Red
Output only (Non-modifiable)	Protected, High, White

Note: Dynamic literals, literals which are displayed by the programs in only certain circumstances, should be displayed as normal literals would be displayed. However, in some cases, where it seems more appropriate, they may be displayed as Output only protected fields. For example, the "MORE..." indicator, a dynamic literal, is displayed as a high intensity protected field.

7.3 Screen Cursor Positioning

When a screen is displayed, the cursor must be positioned in the first modifiable (unprotected) field on the screen.

On screens where there are no modifiable fields, the cursor must be positioned on the screen-id (which is always unprotected, column 2 row 1).

Exceptions may be made to this rule on some special screens or where there is a necessity to design a screen in a fashion that will not permit cursor positioning as described above.

When field level errors occur, the cursor will be positioned in the field of the first error. All fields in error will be highlighted and all the fields that can be modified will be unprotected. The user may then tab the cursor to each of those color-indicated fields that remain in error and correct all the errors before sending the transaction. If the user does not correct all the errors before pressing a function key or the Enter key, the cursor is subsequently positioned at whichever field then becomes the first of the remaining errors. The data on this screen will not be processed or updated until all the fields in error are corrected.

7.4 Screen Processing

If the transaction requested in the tran-id field is not in the user's Security profile, the menu screen of the current transaction will display an error message.

When the data entered on the screen are invalid or the access to the transaction is invalid for any reason, the menu screen, on which the transaction requested appears, is displayed with an appropriate error message.

7.5 Naming Conventions/Rules for the Main Menu Screen

Screen Title: The main menu should state the subsystem followed by "MAIN MENU". For example, "WDTIP MAIN MENU". The following format should be followed while creating the menus:

Serial No   Screen Name   Transaction Code

For example:

1	client inquiry menu	B3C1
2	client eligibility menu	B3C2
3	overpayment menu	B3C3

Section VIII: Report Standards

This section outlines standards for the design and development of the WDTIP reports.

8.1 Report Layout Overview

All reports in the WDTIP system will follow a standard report specification format. This will ensure that all reports throughout WDTIP are consistent.

The standard report specification (see Exhibit 1) identifies the various standards to be adhered to.

These standards pertain to:

- ❑ Format
- ❑ Report Title
- ❑ Column/Row Headings
- ❑ Control Area (if required)
- ❑ Editing features
- ❑ Report sequences
- ❑ Page breaks
- ❑ End of report

8.2 Report Details

The following is a description of the standards for Report layouts in the WDTIP system.

8.2.1 Format

The format of each report layout will be in either Landscape or Portrait and will allow 132 or 80 print positions respectively. The total number of detail lines per page should be no more than 40.

8.2.2 Title

All reports will have a title and description to identify the report. The format of the headings will be as below:

Report Id:	Should be place in upper left corner
Report Title:	Title of WDTIP system
Report Desc. line 1:	Report desc line 1
Report Desc. line 2:	Report desc line 2 (if any)

Report Desc. line 3:	Report desc. line 3 (if any)
Report Desc. line 4:	Report desc line 4 (if any)

There should be a blank line between the report header and the report body.

8.2.3 Column/Rows Headings

All headings are to be centered.

The top right hand corner of each report should include:

- Page number
- Run date (MM/DD/CCYY)
  -The date on which the report is printed
- Run time
- As of date (if required) (MM/DD/CCYY)
  - The effective date for which the report is printed.

The top left-hand corner of each report should include:

- Report ID
- Job ID
- County
- Program

The format of these fields are as indicated in the Report Layout Specification.

8.2.4 Control Area

This should help identify who the data is for, or what subset of data is included on the report. The location should be in the upper left corner, below Job ID.

Example: County: Sacramento
Program: JOBS

8.2.5 Editing Features

The following editing rules are to be followed while printing the body of each report. These editing features pertain to the way in which fields are edited and reported.

<u>Alpha Numeric fields</u>
These fields are to be left justified.
<u>Numeric fields</u>

These fields are to be right justified and leading zeroes should be suppressed. Every 1,000 will require a comma separator.

Negative numeric fields

These fields are to be indicated by a minus (-) floating sign on the left of the field.

Rounding off

All numbers are to be rounded by standard rounding logic as applicable and no more than two decimal places are to be reported unless it is specifically requested.

Codes

Codes should be expanded wherever possible to aid in readability.

8.2.6 Sequences

If the report has specific report sequences, then it is to be indicated in the body of the report. On sequence change, desired totals/subtotals should be printed.

Total fields

Total fields will be indicated with a line demarcation.

OR

```
Example:     CalWORKs recipients
                  CalWORKs total

             Food Stamp recipients
                  Food Stamp total
```

8.2.7 Page Breaks

Headings will be printed on the new page.

Carry forward totals and page break totals should be printed, if required.

8.2.8 End of Report

To indicate the end of report, a label "----END OF REPORT----" is to be printed on the last page of each report. The same label will be used when no input data is present.

8.2.9 Footers and Instructions

Report footers and instructions will not be included as part of Report Layouts.

EXHIBIT 1

REPORT LAYOUT SPECIFICATION

```
         1          2          3          4          5          6          7          8          9          0          1          2          3
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012
01REPORT ID: XXXXXXXX                                    REPORT TITLE                                            PAGE:  ZZ,ZZ9
02JOB ID     :  XXXXXXX                                                                                     RUN DATE:  MM/DD/YYYY
03 COUNTY   :  XXXXXXX                                                                                       RUN TIME:  HH:MN
04 PROGRAM: XXXXXXX                                                                                         AS OF DATE: MM/DD/YYYY
05
06                                                        REPORT DESC 1
07                                                        REPORT DESC 2
08                                                        REPORT DESC 3
09                                                        REPORT DESC 4
10
11
12
13
14
15
16                                                        B O D Y    O F    R E P O R T
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31            NOTE: For Portrait layouts the format remains
32                  the same.  Only the width changes.
33
34
35
36
37
38
39
40
                                                      ----END OF REPORT----
```

Section IX: Unit Testing Standards

This section will outline the methodology to be used by individual developers in unit testing modules. Unit testing is used to observe that the subject unit(s) functions as expected according to design specifications and outputs accurately and with anticipated results. The objective is to establish the successful execution of all logical paths through the program.

9.1 Test Execution

The unit tests are driven by the programmers and are initiated when a unit of code is complete. Unit testing should test all possible conditions, including error handling, controls, conditional statements, loops, flags, switches, etc. Unit testing should also perform testing to exercise the design (black box) and testing to exercise the internal program logic (white box).

Unit testing should include the use of correct (good) and invalid (bad) data. As a practice, unit tests should test the bad data first, and verify the database is not impacted incorrectly. Once verified, the good data should be used in testing (previously tested bad data may be utilized again as appropriate). This will avoid unneeded time and complexity that occurs if one processes good and bad data concurrently.

9.2 Unit Test Practices

Each programmer is expected to follow these high-level unit test practices:

- Develop an informal test plan for each program.
- Meet the objective of unit test - confirm the unit is coded correctly.
- Conduct unit test - Functions are exercised, code is exercised, extremes and boundaries are explored.
- Determine completeness by the removal of all known defects and the programmer feels comfortable with the quality.
- Unit testing incidents are not to be recorded.

A checklist for unit tests will be included in the **Completed Source Modules/Unit Test** deliverable.